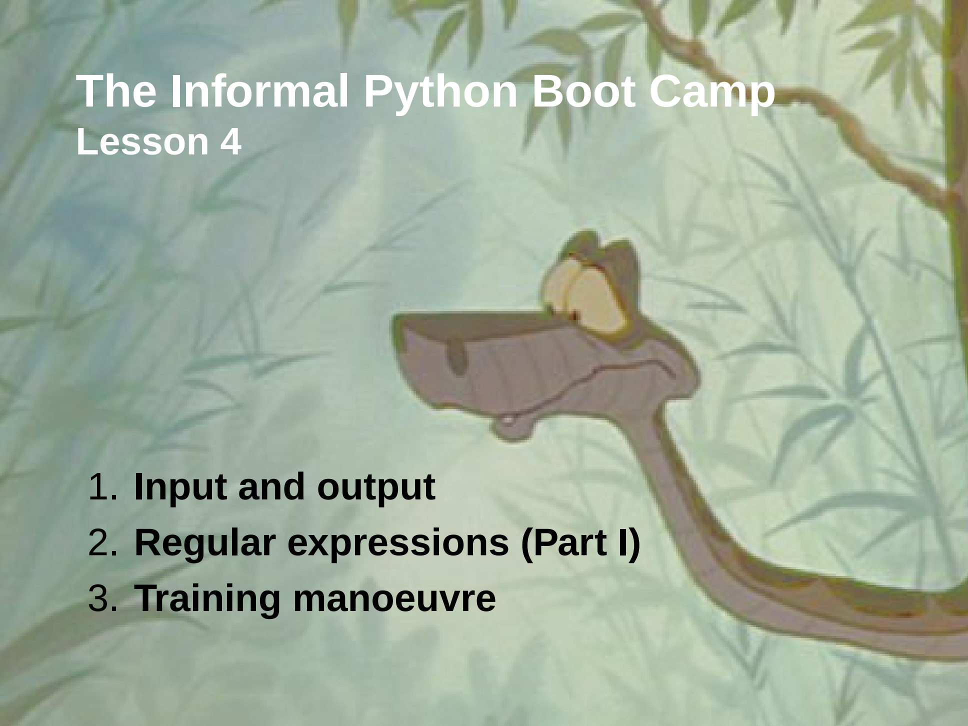
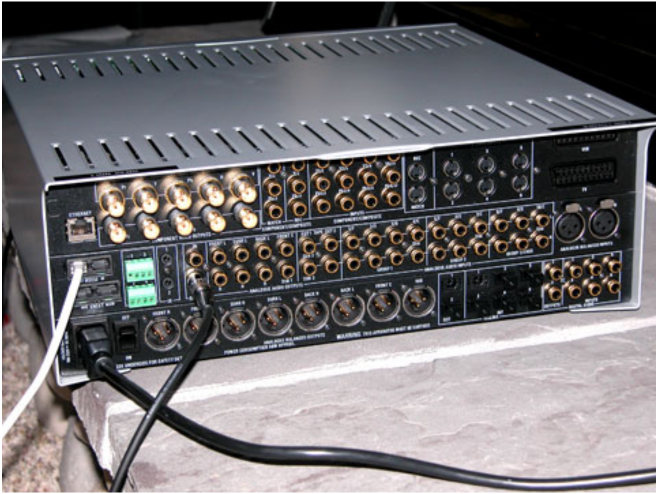


The Informal Python Boot Camp

Lesson 4

1. **Input and output**
2. **Regular expressions (Part I)**
3. **Training manoeuvre**

Input and output



Opening a file

Use `open(filename,mode)` to open a file:

```
f = open(’/tmp/workfile’, ’w’)
```

Some possible modes:

- `r`: Open text file for read.
- `w`: Open text file for write.
- a: Open text file for append.
- `rb`: Open binary file for read.
- `wb`: Open binary file for write.

Open returns a *File Object*.

To close the file, use:

```
f.close()
```

Predefined File Objects

- `sys.stdin`: Standard input
- `sys.stdout`: Standard output
- `sys.stderr`: Standard error

Consequent use of `stderr` for all warnings, status messages and error messages while using `stdout` for any ”real” output is a good practice, because this allows cool piping tricks!

Reading from a File Object

Read a quantity of data from a file:

```
s = f.read( size )  # size: number of bytes to read
```

Read entire file:

```
s = f.read()
```

Read one line from file:

```
s = f.readline()
```

Get all lines of data from the file into a list:

```
list = f.readlines()
```

Iterate over each line in the file:

```
for line in f:
    print line,
```

Writing to a File Object

Write a string to the file:

```
f.write( string )
```

Write several strings to the file:

```
f.writelines( sequence )
```

Flush the internal file buffer:

```
f.flush()
```

Don’t forget to `flush()`, when talking to another program over a pipe and want to have an immediate response!

Example: A primitive cat replacement

```
import sys

if len(sys.argv) > 1:
    for filename in sys.argv[1:]:
        f = open(filename, ’r’)
        for line in f:
            sys.stdout.write(line)

        f.close()
else:
    sys.stdout.writelines( sys.stdin )
```

```
Usage: cat.py [FILE]...
Concatenate FILE(s), or standard input,
to standard output.
```

Inter-process communication with pipes

Read from another program:

```
import os
gz = os.popen(’gunzip -c compressed_file.gz’,’r’)
uncompressed_data = gz.readlines()
gz.close()
```

Write to another program:

```
import os
gz = os.popen(’gzip > compressed_file.gz’,’w’)
gz.write(uncompressed_data)
gz.close()
```

The `pickle` module

Saving of arbitrary python data structures is trivial:

```
>>> import pickle
>>> x = [’a’, [’nested’, ’data’, ’structure’,
         (1, 2, 3)]]

>>> f = open("my_file", "w")
>>> pickle.dump( x, f )
>>> f.close()

>>> f = open("my_file", "r")
>>> x =  pickle.load( f )
>>> f.close()

>>> print x
[’a’, [’nested’, ’data’, ’structure’, (1, 2, 3)]]
```

See also these other modules: `shelve`, `anydbm`, `cPickle`

Regular expressions

```
[-+]?(\d+(\.\d*)?|\.\d+)([eE][-+]?\d+)?
```

Introduction

Regular expressions are a powerful tool to do tasks like:

- Extract values from strings like `'lat=22.5, lon=5'`.
- Remove all HTML formatting from a web page.
- Strip off any filename extensions from a list of filenames.
- Extract all section names from a LaTeX file.
- Check user input to be in a specific format.
- Replace all whitespace sequences in a text with single spaces.

Use them, where string methods like `string.find()` *or* `string.replace()` *don't offer enough flexibility.*

Introductory example

Extract numbers from a string:

```
>>> import re
>>> p = re.compile(’\d+’)
>>> p.findall("""12 drummers drumming,
                 11 pipers piping,
                 10 lords a-leaping""")
[’12’, ’11’, ’10’]
```

String pattern matching with regular expressions

Regular expressions are in very wide use:

- Unix tools: awk, sed, grep, ...
- Editors: emacs, vi, nedit, kate, ...
- Programming languages: perl, ...
- Regex libraries exist for almost any programming language.

String pattern matching with regular expressions

Regex Trivia

- Regular expressions are written in their own language.
- Dialects differ slightly.
- Most newer tools use `perl` style regular expressions.
- Documentation: `http://perldoc.perl.org/perlre.html`
- Book: *Mastering Regular Expressions - by Jeffrey E. F. Friedl*

String pattern matching with regular expressions

Regular expressions in Python

- Python uses `perl` style regular expressions.
- Regular expression are provided through the `re` module.
- Python specific HOWTO:
  `http://www.amk.ca/python/howto/regex/`

Simple patterns

Matching characters:

```
>>> text = "Currywurst, Bratwurst, Wurst"
>>> re.findall( r’Brat’, text )
[’Brat’]
>>> re.findall( r’Tofu’, text )
[]
```

Simple patterns

These characters have a special meaning for the regex:

```
    . ˆ $ * + ? { [ ] \ | ( )
```

Yes, also the dot!

They must be backslash-escaped when searching for them:

```
>>> text = "lesson1.tex lesson1.pdf"
>>> re.findall( r’\.tex’, text )
[’.tex’]
```

Character classes

- Brackets match any of the enclosed characters.

Example 1: `[12]` matches 1 as well as 2.

```
>>> text = "lesson1.tex lesson2.tex"
>>> re.findall( r’lesson[12]’, text )
[’lesson1’, ’lesson2’]
```

Example 2: `[t-z]` matches any character between ’t’ and ’z’.

```
>>> text = "lesson1.tex lesson2.tex"
>>> re.findall( r’[t-z]’, text )
[’t’, ’x’, ’t’, ’x’]
```

Character classes

- Use `[^...]` to match any characters not in the class.

Example: `[^a-z]` matches any non-lowercase letters:

```
>>> text = "lesson1.tex lesson2.tex"
>>> re.findall( r’[^a-z]’, text )
[’1’, ’.’, ’ ’, ’2’, ’.’]
```

Character class shortcuts

- `\d` Matches any decimal digit; equivalent with `[0-9]`.
- `\D` Matches any non-digit character; `[ˆ0-9]`.
- `\s` Matches any whitespace character; `[ \t\n\r\f\v]`.
- `\S` Matches any non-whitespace character; `[ˆ \t\n\r\f\v]`.
- `\w` Matches any alphanumeric character; `[a-zA-Z0-9_]`.
- `\W` Matches any non-alphanumeric character; `[ˆa-zA-Z0-9_]`.

Character class shortcuts

- The dot `.` matches any character.

Repeating things

- Match it one or more times by appending a + to it.
- Match it zero or more times by appending a `*` to it.

```
>>> text = "width=800, height=600"
>>> re.findall( r'\d+', text )
['600', '800']
```

- Match it between `n` and `m` times by appending `{n,m}` to it.

Optional parts of a pattern

- Use parantheses to group a part of a pattern.
- Append a questionmark to an optional part of a pattern.

Example: `r'((curry)?brat)?wurst'` matches any of `'wurst'`, `'bratwurst'` and `'currybratwurst'`.

By default patterns are *greedy*, so the longest possibe match will win.

Alternation

- Use the ”or” operator | to specify alternatives.

Example: `r’(curry|brat)wurst’`
matches `’bratwurst’` as well as `’currywurst’`.

Anchors

- ˆ matches the beginning of the string.
- $ matches the end of the string.

Example:

```
>>> text = ’From Here to Eternity’
>>> re.findall( r’ˆFrom’, text )
[’From’]

>>> text = ’Reciting From Memory’
>>> re.findall( r’ˆFrom’, text )
[]
```

Compiling regular expressions

```
>>> import re
>>> re.findall(’\d+’, """12 drummers drumming,
                         11 pipers piping,
                         10 lords a-leaping""")
[’12’, ’11’, ’10’]
```

has the same effect as

```
>>> import re
>>> p = re.compile(’\d+’)
>>> p.findall("""12 drummers drumming,
                 11 pipers piping,
                 10 lords a-leaping""")
[’12’, ’11’, ’10’]
```

but the latter may be faster, if evaluated several times.

That’s enough for today!

Next week, we’ll see how to gain even more power with regular expressions:

- Search and replace
- Split
- Capture parts of a match

Things to remember from today's lesson

- Files and pipes behave in the same way.
- If it seems complicated, maybe it's simple with a regex.
- Python regex HOWTO:
  `http://www.amk.ca/python/howto/regex/`

Keep these pages open, when programming in Python:

- Tutorial:
  `http://docs.python.org/tut/`
- Library Reference:
  `http://docs.python.org/lib/lib.html`

Training manoeuvre

Problem: parsing tabular data with blank lines and headers

```
% slinktool  -u  -S ’GR_BSEG:BHZ.D’ ersn12.szgrf.bgr.de
GR_BSEG_BHZ, 412 samples, 20 Hz, 2007,284,13:42:37.574707 (latency ˜3.4 sec)
       656         715         692         612         643         705
       677         686         710         732         783         726
...

GR_BSEG_BHZ, 412 samples, 20 Hz, 2007,284,13:42:58.174707 (latency ˜4.0 sec)
       745         779         720         731         770         760
       775         784         794         740         733         758
...
```

Write a script which

- Reads this stuff.
- Puts the data values to stdout.
- One value per line.
- And dumps header information to stderr.